5. Finding Partitions Without Expert Knowledge

From **"**Verification, Validation, and Evaluation of Expert Systems, Volume I"

This chapter presents techniques for partitioning large expert systems when expert knowledge is unavailable.

Introduction

Generally, it is best to partition a knowledge base using expert knowledge, resulting in a knowledge base that reflects the expert's conception of the knowledge domain. This, in turn, facilitates communication with the expert, and later maintenance of the knowledge base. Chapter 7, "Knowledge Modeling", presents techniques for partitioning using expert knowledge.

Sometimes, however, it is not possible to obtain expert insight into a knowledge base. In this case functions and incidence matrices can be extracted from the knowledge base, and the information contained therein used to partition the knowledge base.

Functions

Expert Systems are Mathematical Functions

Expert systems are, among other things, complicated functions in the mathematical sense of function. [By definition, a function is a set F of ordered pairs, such that if (a,b) and (c,d) are in F, and a = c, then b = d.] Less formally, a function is a single-valued mapping from an input space (called the domain) to an output space (called the range); i.e., there is only one value of the function for each point in the input space. For example, KB1 is a function that for each set of user data (i.e., amount of savings, personal property, etc.) assigns a type of investment.

The input variables to an expert system viewed as a function are the variables that are not computed inside the expert system, but are asked the of user or looked up in a data base. Variables that are inferred by rules or computed by functions in the knowledge base are not input variables. In KB1, for example, purchase of lottery tickets and ownership of boats and luxury cars are input variables, while risk tolerance and discretionary income are not. Tolerance and discretionary income, however, are inputs to the investment subsystem of KB1.

Propositions that are possible conclusions of the expert system are Boolean output variables of the expert system. Numerical or enumerated variables that are considered outputs of the expert system are also output variables. When viewed as a function the value of an expert system is a vector of these individual output variables.

Partitioning Functions into Compositions of Simpler Functions

Functions can be written as compositions of simpler functions. For expert systems, two of the important relations that build more complex functions from simpler ones are Cartesian product and function composition.

Cartesian Product

Suppose that an expert system made two different kinds of recommendations, e.g., a traffic management system that both set the timing of lights and controlled access to exit ramps. This expert system could be considered as a function E that computed light timing and on ramp access from certain inputs, e.g.:

E(inputs) = (timings, access).

E could be split into two expert systems that computed these results separately:

E = (timings(inputs), access(inputs)) (5.1).

While some of the inputs and intermediate conclusions might appear in both subsystems, (5.1) decomposes E into two subsystems using the Cartesian product operation. The Cartesian product operation in this case takes the two separate conclusions, timings(inputs) and access(inputs) and builds the conclusions of E:

(timings(inputs), access(inputs))

by putting the separate conclusions of the subsystems together in a fixed, predetermined order.

More generally, if:

(y1,...,ym) = f(w1,...,wk),

(z1...,zq) = g(x1,...,xn,),

then:

(y1,...,ym, z1...,zq) = f(w1,...,wk) X g(x1,...,xn,),

where X is the Cartesian product operator.

Applied to expert systems, this result means that if there is an expert system where input Ws are used to compute the conclusion Ys, and the Xs are used to compute the Zs, the system can be partitioned into subsystems:

(y1,...,ym) = f(w1,...,wk),

(z1...,zq) = g(x1,...,xn,),

and the results concatenated together.

Function Composition

Function composition uses the results of an earlier function A as the inputs to a later function B to compute a single overall function C. This overall function is the result of :

1. Starting with the inputs to A.

2. Applying the function A to these inputs.

3. Applying B to the results of Step 2.

4. Using the results of step 3 as the value of C.

In the Pavement Maintenance Expert System (PAMEX), for example, various data items are used to compute the "Pavement Serviceability Index" (PSI) and other measures of pavement life. The PSI and other similar parameters are then fed into a follow-up set of rules that choose appropriate maintenance procedures. PAMEX can be considered as a composition of the subsystem that computes indices with the subsystem that uses these to compute appropriate maintenance procedures.

In mathematical notation, suppose the output of an expert system depends on a set of variables, y1,...ym, i.e.:

E = f(y1,...,ym)

In addition, suppose each of the y's is a function of some other variables, i.e.,:

yi = gi(x1,...,xmi)

Then E = f(g1(x11,...,x1m),

g2(x21,...,x2m),

....

gn(xm1,...,xnm))

i.e., the expert system E is the result of applying the function f to the result of applying Gs to the input variables.

Note that which variables are functions of which others are properties of the expert system. This means that a function implemented by an expert system can not be arbitrarily rewritten as the composition of simpler functions. Instead, the choice of simpler functions is motivated by:

- Which variables are functions of which other ones in the expert system knowledge base.
- Which rewriting of the function computed by an expert system as the composition of functions reduces the size of the VV&E problem.

For KB1, investment is a composition of an investment function with risk tolerance and discretionary income functions:

- investment(risk_tolerance("lottery tickets", "stock ownership"),
- discretionary_income("boat", "luxury car")).

Dependency Relations

To find the functions embedded in a knowledge base, it is helpful to compute the dependency relation among variables.

Immediate Dependency Relation

The first step is to compute the immediate dependency relation. If X1 and X2 are variables in the knowledge base, X2 is immediately dependent on X1, if and only if, the following are true:

- X1 appears in an expression that computes X2.
- X1 appears in the if part of a rule that sets or concludes X2.
- X1 is an input to a function that computes X2.

The table below shows the immediate dependency relation for Knowledge Base 1. A1 appears in cell (I,Jj), if and only if, variable J is immediately dependent on variable I.

The immediate dependency relation for Knowledge Base 1 is shown in table 5.1.

Table 5.1: Immediate Dependency Relation for KB1

immediate dependency	LC	B	S	LT	DI	RT	INV
luxury car (LC)	0	0	0	0	1	0	0
boat (B)	0	0	0	0	1	0	0
stocks (S)	0	0	0	0	0	1	0
lottery tickets (LT)	0	0	0	0	0	1	0
discretionary income (DI)	0	0	0	0	0	0	1
risk tolerance (RT)	0	0	0	0	0	0	1
investment (INV)	0	0	0	0	0	0	0

The immediate dependency relation shows which variables influence the value of other variables through one level of computation (one rule inference or function computation) in the expert system.

Computing the Immediate Dependency Matrix

The immediate dependency matrix can be computed by syntactic inspection of the source code (including both rules and procedures) of the expert system knowledge base. Although the underlying computation is basically the same, the computation can be described either as a database or as a sparse matrix computation.

Database Description of Immediate Dependency Computation

The immediate dependency matrix can be constructed directly as follows. In this construction, the matrix is represented by a relation with 2 colums:

- Column 1: A variable that affects another variable.
- Column 2: A variable that is affected by another variable.

Each row in the table represents a pair of variables such that the first affects the second directly in some rule or function.

Start with an empty database.

For each rule or function in the knowledge base, find all pairs (x,y) such that x is an input and y an output of the rule or function. Put each such pair in the database.

It is also possible to construct the data base as the composition of two simpler tables:

An input table:

- Column 1: An input variable.
- Column 2: A rule or function in the knowledge base.

A row (x,f) appears in this table when a variable x is an input to a rule or function f.

An output table:

- Column 1: An output variable.
- Column 2: A rule or function in the knowledge base.

A row (x,f) appears in this table when a variable x is an output to a rule or function f.

Now by applying the following join operation to the tables, build atable where:

- Column 1 is an input variable.

- Column 2 is an output variable.

There is a row for each variable pair (x,y) such that for some f, (x,f) is in table 1 and (y,f) in table 2.

Sparse Matrix Description of Immediate Dependency Computation

The relation between input and output variables describes a aparse matrix representing the immediate dependency relation. The rows and columns are indexed by variables. A 1 appears for the matrix position described by each row in the table constructed in the preceeding section, and a 0 appears for all other matrix positions. By the definition of the immediate dependency relation, this sparse matrix represents that relation.

The join-based computation described above can be written using sparse matrices as follow:

1. Construct input and output matrices:

   The input matrix is based on table 1. The rows are indexed by variables and the columns by functions and rules. A 1 appears when a variable is an input to a rule or function. Zeros fill the other matrix positions.

   The Output matrix is based on table 2, but is the transpose of the matrix that directly represents table 2. The rows are indexed by functions and rules. The columns are indexed by variables. A 1 appears when a variable is an output of a rule or function. Zeros fill the other matrix positions.

2. Compute the product of the input matrix by the output matrix.

3. Booleanize the product matrix, i.e. replace all non-zero entries by 1s.

This product matix has a 1 at position (x,y) whenever the product has a non-zero, i.e. when there is a rule or function f where x is an input to f and f has y as an output.

An Example

Dependency Relations of Rules on Variables in Knowledge Base 1

In KB1, all atomic formulas set by the knowledge base are of the form:

VARIABLE = VALUE

When this is the case, the immediate dependency of variables and rules is sufficient to obtain the dependency among variables. Table 5.2 shows how variables influence rules.

Table 5.2: How Variables Influence Rules

	R1	R2	R3	R4	R5	R6
LC					1	1
B					1	1
S			1	1		
LT			1	1		
DI	1	1				
RT	1	1				
INV						

Dependency Relations of Variables on Rules in Knowledge Base 1

Table 5.3 shows how rules influence variables.

Table 5.3: How Rules Influence Variables

	LC	B	S	LT	DI	RT	INV
R1							1
R2							1
R3						1	
R4						1	
R5					1		
R6					1		

Dependency Relations of Variables on Variables in Knowledge Base 1

Multiplying A*B creates the matrix showing how each variable influences others. Positive numbers in cell (R,C) indicate that the variable in row R influences the variable in column C. Making this into a Boolean matrix yields the immediate dependency matrix for variables in KB1.

Table 5.4 shows the immediate dependency matrix for KB1.

Table 5.4: Immediate Dependency Matrix for KB1

	LC	B	S	LT	DI	RT	INV
LC	0	0	0	0	2	0	0
B	0	0	0	0	2	0	0
S	0	0	0	0	0	2	0
LT	0	0	0	0	0	2	0
DI	0	0	0	0	0	0	2
RT	0	0	0	0	0	0	2
INV	0	0	0	0	0	0	0

Using the extended immediate dependency relation R just defined, the user can compute a sub-knowledge-base that is sufficient to compute a set of variables. Let SO be a set of output variables for a function f, chosen as discussed in the previous section. Let RR be either one of the R *a n or the relation R *d. Then the sub-knowledge base that computes f is defined by:

x is in Sub_KB(f) iff x RR y for some y in SO.

Operations on Relations

Using the immediate dependency relation, one may compute the influences of variables through any number of levels of inference or function computation and composition. This requires union and composition relations defined as follows:

Relation: A relation is, from a mathematical standpoint, a set of ordered pairs.

For example, the immediate dependency relation is shown as an ordered pair in figure 5.1:

{(LC,DI), (B,DI), (S,RT), (LT,RT), (DI,INV), (RT,INV)}

A pair (x,y) appears in the immediate dependency relation if and only if x influences the value of y.

Figure 5.1: Immediate Dependency Relation as Ordered Pairs

Domain: If R is a relation {x| for some y, xRy} is the *domain* of R. Some examples of domains are shown in figure 5.2.

Domain of the investment subsystem of KB1:

{ ("discretionary income" = yes, "risk tolerance" = high),

("discretionary income" = no, "risk tolerance" = high),

("discretionary income" = yes, "risk tolerance" = low),

("discretionary income" = no , "risk tolerance" = low)}

Domain of the immediate dependency relation for KB1:

{luxury car, boat, stocks, lottery tickets, discretionary

tolerance, risk tolerance, investment}

Figure 5.2: Examples of Domains

Range: {y| for some x, xRy} is the range of r. For example, the range of the investment subsystem of KB1 is { stocks, savings account}; the range of the immediate dependency relation is {0, 1}.

Composition: If R1 and R2 are relations, the relation (R1 o R2) is defined as follows: x (R1 o R2) z if and only if there is a y such that x R1 y and y R2 z.

For example, the composition of the immediate dependency relation of KB1 with itself is:

{(LC,INV), (B,INV), (S,INV), (LT,INV)}.

For an immediate dependency relation R among the variables of an expert system, (x,z) is in RoR if and only if there is a y such that (x,y) and (y,z) are in R; i.e., there is a variable y such that x influences y and y influences z. In other words, RoR shows the variables that indirectly influence another variable acting through a single intermediate variable.

Matrix representation: When range(R1) = domain(R2)

the composition operation R1 o R2 can be computed by matrix multiplication. A relation R is represented by a matrix M = {m(i,j)} if and only if:

m(i,j) = 1 iff x R y where x is variable i and y is variable j

m(i,j) = 0 otherwise.

Table 6.1 shows the immediate dependency relation in matrix form.

If Mi represents Ri, B(M1 o M2) represents R1 o R2, where:

M1 o M2 represents matrix product of M1 and M2.

B(M) = {bm(i,j)} represents the Boolean operation on matrices, i.e.,

bm(i,j) = 1 iff m(i,j) != 0

bm(i,j) = 0 iff m(i,j) = 0.

Theorem 5.1: If R1 and R2 are immediate dependency matrices, B(M1 o M2) represents R1 o R2 when M1 represents R1 and M2 represents R2.

This theorem says that the representation of the indirect dependency relation with one intermediate variable can be computed by Booleanizing the matrix product of the immediate dependency matrix with itself.

Proof: Let M be the matrix that represents R1 o R2, based on a numbering of the relevant variables v1,...vn. The (i,j) entry of M is 1 if and only if vi influences vj. This means that there two sets of inputs where the vi's differ, and also where the results of applying (R1 o R2) to these inputs differ. On these two inputs, one of the inputs to R2 must vary on the two inputs; if no input to R2 varied, the output would also not vary on the two inputs.

Since at least one input variable to R2 varies when vi varies, let vk be such an input to R2. Since vk varies when vi varies, R1(i,k) = 1. Likewise, since vj varies when vk varies, R2(k,j) = 1. This means that:

the kth entry of row i = 1

the kth entry of column j = 1.

As a result, kth summand in the inner product:

(Row i of M1) * (column j of M2) (5.2)

is 1. Since all entries of M1 and M2 are non-negative, the Cartesian product (6.2) is non-zero. This means that (M1 o M2) has a non-zero (i,j) entry, so B(M1 o M2)(i,j) = 1. The result is that everywhere M is 1, B(M1 o M2) is also 1.

Now let (m,n) be a location in B(M1 o M2) which is 1. This will be true only if the (m,n) entry of M1 o M2 is non-zero. Since all entries of M1 and M2 are non-negative, (M1 o M2)(m,n) > 0. This entry of M1 o M2 is the inner product:

(row m of M1) * (column n of M2)

so the inner product is positive. This is possible only if there is a k so that the kth entry in each of these vectors is non-zero. This means that for some k, the kth entry of row m of M1 and the kth entry of column n of M2 are both 1, i.e.,:

M1(m,k)=1

M2(k,n)=1.

This means that vm influences vk and vk influences vn. Therefore, vm influences vn, showing that M, the representation of (R1 o R2), has a 1 wherever B(M1 o M2) has a 1.

Combined with the earlier result, it is evident that the two matrices M and B(M1 o M2) have the same set of 1's. Since both matrices have only 1 and 0 entries, the matrices are equal.

For example, in KB 1, B influences DI, as indicated by the 1 in the (B,DI) entry of the immediate dependency relation of KB1. In table 6.1, this appears in the (2,5) location. Likewise, DI influences INV, and the (5,7) entry of the table is 1, meaning that multiplying the table by itself, when the inner product of row 2 by column 7 is computed, the 1's in position 5 cause the inner product to be non-zero. This represents the fact that variable 2 (B) influences INV, variable 7, through the intermediary of variable 5, VI.

Table 5.5 shows the matrix product of the immediate dependency relation by itself. In this case, it is also the Boolean composition operation.

Table 5.5: Matrix Product of the Dependency Relation by Itself

immediate dependency	LC	B	S	LT	DI	RT	INV
luxury car (LC)	0	0	0	0	0	0	1
boat (B)	0	0	0	0	0	0	1
stocks (S)	0	0	0	0	0	0	1
lottery tickets (LT)	0	0	0	0	0	0	1
discretionary income (DI)	0	0	0	0	0	0	0
risk tolerance (RT)	0	0	0	0	0	0	0
investment (INV)	0	0	0	0	0	0	0

Power: If R is a relation,

R**1 = R

R**(n+1) = R o (R**n).

The power relation finds those variables which influence a variable through a chain of intermediate variables of some particular length. For R**n the chain of intermediate variables is of length n-1.

If M represents R and M**n is the product of n Ms, then B(M**n) represents R**n.

The previous table shows R**2 when R is the immediate dependency relation. Higher powers of the immediate dependency relation are empty (all zeros in the matrix representation).

Theorem 5.2: M**n represents the indirect influence of variables with n-1 intermediate variables.

Proof: Theorem 5.2 follows from Theorem 5.1 by mathematical induction.

Union: If R1 and R2 are relations with the same domain and range, the relation (R1 U R2) is the relation such that x (R1 U R2) y iff x R1 y or x R2 y.

The union and composition operations are used to build relations about dependency through multiple levels of inference. For example, if x D2 y, if and only if x influences y, directly or through an intermediate variable, D2 = D U D o D, where D is the intermediate dependency relation and o is the composition operation.

Theorem 5.3: If Mi represents Ri, B(M1+M2) represents R1 U R2.

Proof: B(M1+M2)(i,j) = 1 iff M1(i,j) or M2(i,j). Iff x is the ith variable and y is the jth variable, M1(i,j) or M2(i,j) iff x R1 y or x R2 y, i.e.

x (R1 U R2) y.

Figure 5.2 represents:

R U (R**2)

where R is the immediate dependency relation of KB1.

Accumulation: The accumulation operator R *a n is defined as follows:

R *a 1 = R

R *a (n+1) = (R *a n) U (R ** (n+1))

The accumulation R *a n of a relation finds all the variables that influence a variable through a chain of n-1 or fewer intermediate variables.

Theorem 5.4: R *a n represents the dependency relation between n-1 or fewer intermediate variables. If M represents R, B(M *a n) represents R *a n.

Proof: This follows from Theorems 5.2 and 5.3.

Dependency: The relations { lim R *a n } form an increasing sequence of relations, i.e., if (x,y) is in *a n, (x,y) is in *a m for m >= n. Therefore, the limit of this sequence as n --> infinity exists, and is equal to the union of the R *a n for all n. This limit will be called R*d.

Define the dependency relation D(R) as follows: x D(R) y iff the variable x influences the variable y. It is only possible for x to influence y if there is some (possibly empty) chain of intermediate, e.g., x, z1, ..., zn, y such that each variable influences its successor, i.e., each successive pair of variables is in the relation R. However, then x R**(n+1) y, so x (R *a n) y,

so x R*d y, and D(R) <= R*d.

However, if x R*d y, for some n, (x,y) R*a m for m > n (by definition of limit). Pick an m0 > n. Then x R*a m0 y, so for some m1 <= m0,

x R**(m1) y. Then there is a chain of m1+1 intermediate variables, z1,...zm1+1 such that x,z1,...,zm1+1,y is a sequence in which successive variables are in R, and R*d < D(R).

Combining this with the previous result proves theorem 5.5.

Theorem 5.5: The limit R*d of the accumulation relations represents the dependency relation D(R).

Since both the sequences {B(M*n)} and {R*n} are monotone increasing and have only a finite number of possible values, each of these sequences is eventually constant. That constant is the limit of the sequence. Pick an n0 great enough so that each sequence has reached its limit. By Theorem 5.4, B(M *n0) represents R *n0 where M represents R. Since equal matricies represent equal relations, the limits can be substituted in this "represents" relation, proving

Theorem 5.6: The matrix lim(n->infinity)(B(M *a n)) represents D.

The dependency relation represents the relation that is true for all variables that influence a given variable, and false otherwise. Figure 5.2 is the accumulation of the immediate dependency relation of KB1. An entry in the table is 1 iff the variable on the right is dependent on a variable on the left.

To compute the dependency relation from the immediate dependency relation:

- Compute in sequence each R *a n.
- When the R *a n no longer change, the current R *a n is the dependency relation R*d.

Table 5.6. shows the dependency relation of the immediate dependency relation of Knowledge Base 1.

Table 5.6: Immediate Dependency Relation of KB1

	LC	B	S	LT	DI	RT	INV
luxury car (LC)	0	0	0	0	1	0	1
boat (B)	0	0	0	0	1	0	1
stocks (S)	0	0	0	0	0	1	1
lottery tickets (LT)	0	0	0	0	0	1	1
discretionary income (DI)	0	0	0	0	0	0	1
risk tolerance (RT)	0	0	0	0	0	0	1
investment (INV)	0	0	0	0	0	0	0

Finding Functions in a Knowledge Base

To carry out a partition of a knowledge base based on function composition, it is necessary to find functions embedded in the knowledge base. In particular, the goal is to find subsets SI and SO of the knowledge base variables such that the:

- Values of SO are a function of the inputs in SI.
- Variables in SI are used at most infrequently outside this function.

Choosing the Output and Input Variables of a Function

Each column vector in the dependency relation matrix shows which variables influence each other. For example, the first 4 columns of the dependency matrix for KB 1 are all 0s, because these are input variables and are not influenced by any other variables in the KB. Discretionary income (DI) has 1's for the two variables that influence it, namely the boat and luxury car. Investment has nearly all 1's, because all variables except itself influence its value.

To find the set of variables whose Cartesian product will be the output of a function in the KB, cluster via high correlation the column vectors in the table. The clusters should be performed in such a way that all members of a cluster are highly correlated with each other, indicating that all the variables computed by a function use about the same set of input variables.

The variable clusters of the dependency relation of the immediate dependency relation of Knowledge Base 1 are:

{luxury car, boat}

{stocks, lottery tickets}

{discretionary income, risk tolerance}

{investment}

Once a set of output variables has been chosen, the set of input variables for the function consists of the union of all variables for each member of the output variable set. Table 5.7 shows variable clusters of the dependency relation of KB1.

Table 5.7: Variable Clusters of the Dependency Relation of KB1

VARIABLE CLUSTER	INPUT VARIABLES
{LC, B, S, LT}	none
{DI}	{LC,B}
{RT}	{LT,S}
{INV}	{DI,RT}

Finding the Knowledge Base that Computes a Function

In the previous section, the input and output variables were computed for a set of functions that partition the knowledge base. Table 5.4 illustrates this partitioning for knowledge base 1.

Given the input and output variables for a function, the subset of rules and functions in the knowledge base used to compute that function can be found as follows. Note that the input and output matrices from which the immediate dependency relation is computed are used in this computation. Refer to **Computing the Immediate Dependency Relation** for details about computing these matrices.

1. Start with the output variables of the function. Set the current unprocessed output variables to the set of output variables. Start with an empty set of rules and KB functions in the KB subset implementing the function; call the set of implementing and rules IMP.

2. For each current unprocessed output variable y, and each function or rule f which has y as an output, add f to IMP. Remove y from the set of unprocessed output variables.

3. For each f added to IMP, examine all x such that x is an input to f. If x is not an input to the function for which a KB is being computed, add x to the set of unprocessed output variables.

4. Continue this process until the set of unprocessed output variables is empty.

Hoffman Regions

For logical completeness and consistency of an expert system, an important concept is the Hoffman regions (suggested by Roger Hoffman of FHWA). If V1...Vn are the variables of a knowledge base, with domains D1...Dn respectively, a Hoffman region is a maximal subset of the input space, the Cartesian product D1x...xDn, on which each atomic formula in the knowledge base has a single truth value. For any knowledge base, there is a unique set of Hoffman regions that cover and partition the input space.

A run of an expert system is completely determined by the values of the atomic formulas that appear in the KB rules. Provided that the expert system does not use external numerical software, there is no

need to run two different test cases that evaluate the same on all the atomic formulas. If two different test cases evaluate some atomic formula differently, however, the firing of some rule, and hence the results of the expert system, may differ between the two test cases. Therefore, the set of test cases that must be tested are in 1-to-1 correspondence with the regions where all the atomic formulas have the same value. These regions where the atomic formulas are the same are called Hoffman regions.

Each point in input space determines truth values for each of the atomic formulas in the knowledge base. A relation H(P1,P2) can be defined on input point spaces as follows: H(P1,P2) is true if and only if P1 and P2 determine the same set of atomic formula truth values for all atomic formulas in the KB. H so defined is an equivalence relation, and partitions the input space into mutually disjointed regions that cover the input space.

It is generally not possible to find simple, exact descriptions for all the Hoffman regions when a knowledge base contains atomic formulas that contain several variables, e.g., exp(X)<Y^3. It is possible, however, to find an approximate set of Hoffman regions of descriptions such that:

- Every Hoffman region is in the approximate set of Hoffman regions.
- A member of the approximate set of Hoffman regions is either a Hoffman region, or is the empty set, i.e. is an empty region of input space.

The set of possible Hoffman descriptions D can be computed as follows:

- For atomic formulas containing two or more variables, the Hoffman regions of these atomic formulas are TRUE and FALSE.
- Sort all the atomic formulas containing only one variable into subsets, putting all the formulas containing the same variable together.
- Normalize formulas containing relation operators so that the variable appears on the left.
- Lexically sort the formulas for each variable as follows:
  - The major sort is by the right side of the formula.
  - The minor sort is by relational operator, where the relation operators in ascending order are: <, <=, =, >=, >.
- Create a set of intervals for each numerical variable that:
  - Cover the real line, or at least the possible domain of the variable.
  - For all points in any interval, the truth values of the atomic predicates (of that single variable) are the same.
  - The intervals are maximal, given the truth value constraint.
- For each string variable, let the Hoffman regions be the list of values that appear in the KB.
- Let the Hoffman regions of the KB as a whole be the Cartesian product of the Hoffman regions for the individual variables.

Note that in KB's with atomic formulas with more than one variable, the use of TRUE or FALSE as the Hoffman regions is a compromise to avoid having to decide exactly when combinations of these formulas are true. This means that some Hoffman regions may be unsatisfiable. Therefore, if

exhaustive testing shows an inconsistency in some Hoffman region which is partly defined by atomic formulas of more than one variable, there are two possibilities:

- The Hoffman region is unsatisfiable, so the expert system is OK.
- The Hoffman region is satisfiable, and the expert system has an inconsistency.

If a Hoffman region is found where the expert system is inconsistent, it should be determined whether the Hoffman region is satisfiable. Table 5.8 illustrate this concept.

Table 5.8: Hoffman Regions for KB1

LC=yes B=yes LT=yes S=yes	LC=yes B=yes LT=yes S=no	LC=yes B=yes LT=no S=yes	LC=yes B=yes LT=no S=no
LC=no B=yes LT=yes S=yes	LC=no B=yes LT=yes S=no	LC=no B=yes LT=no S=yes	LC=no B=yes LT=no S=no
LC=yes B=no LT=yes S=yes	LC=yes B=no LT=yes S=no	LC=yes B=no LT=no S=yes	LC=yes B=no LT=no S=no
LC=no B=no LT=yes S=yes	LC=no B=no LT=yes S=no	LC=no B=no LT=no S=yes	LC=no B=no LT=no S=no

When is a Partitioning Advantageous

Let CH(KB0) be the cardinality of the Hoffman region set of knowledge base KB0. The worst case in proving a result on a knowledge base KB with sub-KB KB1 is, using the result of the previous section, CH(KB1) + CH(~KB1). If this number is significantly smaller than CH(KB), the partitioning pays off in reducing the size of a VV&E problem.

Hoffman Regions of Partitioned KB1

The KB can be split into the following pieces:

- **Final conclusion KB:** This contains rules 1 and 2, and determines the type of investment.
- **Risk tolerance KB:** This contains rules 3 and 4, and determines the comfort level of the client regarding risk.
- **Discretionary income KB:** This contains rules 5 and 6, and determines whether the client has discretionary income.

Each of these KB's has two input variables each with two values, or four Hoffman regions. Therefore the total number of Hoffman regions after partitioning is twelve, a 25 percent reduction. A greater reduction is found in many larger knowledge bases.